Model properties for efficient synthesis of nonblocking modular supervisors

Control Engineering Practice 112 (2021) 104830

Contents lists available at ScienceDirect

Control Engineering Practice

journal homepage: www.elsevier.com/locate/conengprac

Model properties for efficient synthesis of nonblocking modular supervisors

Martijn Goorden [a,*], Joanna van de Mortel-Fronczak [a], Michel Reniers [a], Martin Fabian [b], Wan Fokkink [c,a], Jacobus Rooda [a]

[a] Department of Mechanical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands
[b] Department of Electrical Engineering, Chalmers University of Technology, Göteborg, Sweden
[c] Department of Computer Science, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

ARTICLE INFO

Keywords:
Supervisory control
Finite automata
Directed graph

ABSTRACT

Supervisory control theory provides means to synthesize supervisors for systems with discrete-event behavior from models of the uncontrolled plant and of the control requirements. The applicability of supervisory control theory often fails due to a lack of scalability of the algorithms. This paper proposes a format for the requirements and a method to ensure that the crucial properties of controllability and nonblockingness directly hold, thus avoiding the most computationally expensive parts of synthesis. The method consists of creating a control problem dependency graph and verifying whether it is acyclic. Vertices of the graph are modular plant components, and edges are derived from the requirements. In case of a cyclic graph, potential blocking issues can be localized, so that the original control problem can be reduced to only synthesizing supervisors for smaller partial control problems. The strength of the method is illustrated on two case studies: a production line and a roadway tunnel.

1. Introduction

The design of supervisors for systems with discrete-event behavior has become a challenge as they comprise growing numbers of components to control and of functions to fulfill, at the same time being subject to market demands requiring verified safety, decreased costs, and decreased time-to-market. Model-based systems engineering methodologies can help in overcoming these difficulties.

The supervisory control theory of Ramadge and Wonham (1987, 1989) provides means to synthesize supervisors from a model of the uncontrolled plant and a model of the control requirements. This synthesis step, in which controllable events that lead to violations of requirements are disabled, guarantees by construction that the closed-loop behavior of the supervisor and the plant adheres to all requirements, is controllable, nonblocking, and maximally permissive. In this context, controllable means that only controllable events are disabled, and nonblocking that special (marked) system states remain reachable. That the supervisor is maximally permissive means that it restricts the controlled behavior as little as absolutely necessary to fulfill the other properties.

It has been shown in Gohari and Wonham (2000) that synthesis is NP-hard, which means that any algorithm computing a supervisor will in the worst case have exponential time and memory complexity. Thus, supervisor synthesis is a tough problem. However, as was also noted by Gohari and Wonham (2000), by observing real-world problems more closely one could discover instances of supervisory control problems that are computationally easier. No suggestions were included in that paper of what these instances might be or how to find them, though.

Analyzing a number of models of industrial-size applications, including the recently published (Moormann et al., 2020; Reijnen et al., 2018, 2017, 2020), one discovers that the synthesized supervisors do not impose any additional restrictions on the system, i.e., the provided set of requirement models is already sufficient to control the plant such that the closed-loop behavior is controllable, nonblocking, and maximally permissive. That is, requirements do not disable uncontrollable events and do not violate nonblocking. Therefore, time and computing resources could have been saved, as the synthesis step turned out to be unnecessary for these cases.

When developing the kind of large infrastructural systems considered in e.g. Moormann et al. (2020) and Reijnen et al. (2017, 2020), due to their safety-criticality, engineers tend to follow a strictly structured development process based on failure mode analysis (Stamatis, 2003). This process includes decomposing the system into components, resulting in a modular plant where no events are shared between components. Furthermore, requirement specifications are formulated in a specific way, where an actuator action is guarded by requirements on the states of other components, typically sensors but also other actuators. So the requirements models are formulated as state-based expressions (Ma & Wonham, 2005; Markovski et al., 2010). As shown

* Corresponding author.
E-mail addresses: m.a.goorden@tue.nl (M. Goorden), j.m.v.d.mortel@tue.nl (J. van de Mortel-Fronczak), m.a.reniers@tue.nl (M. Reniers), fabian@chalmers.se (M. Fabian), w.j.fokkink@vu.nl (W. Fokkink), j.e.rooda@tue.nl (J. Rooda).

https://doi.org/10.1016/j.conengprac.2021.104830
Received 4 September 2020; Received in revised form 16 April 2021; Accepted 17 April 2021
Available online xxxx

in this paper, this specific formulation is beneficial for supervisor synthesis: it can be determined beforehand that synthesis will not impose additional restrictions on the system. Thus, engineers of safety-critical systems working with the specific formulation obtain a powerful tool to gain confidence in the obtained supervisor. This is a crucial aspect for the supervisory control theory, which after 30 years of academic research with impressive results, still has not gained wide-spread industrial acceptance.

This paper takes a different approach compared to an often used methodology in supervisory control synthesis, where particular structures of systems are used to ease synthesis and which are applicable to any given discrete-event system model. Examples of such methods include local modular synthesis (de Queiroz & Cury, 2000), incremental synthesis (Brandin et al., 2004), compositional synthesis (Mohajerani et al., 2014), and coordination control (Komenda et al., 2014). Experimenting with applying several of these synthesis methods directly on the full models of Moormann et al. (2020) and Reijnen et al. (2017) shows that applying the wrong algorithm is fatal in the sense of running out of memory. Thus, knowing beforehand that synthesis is not necessary will save computational time and effort.

This paper brings three contributions, all aimed at easing the synthesis effort necessary for large-scale industrial supervisory control problems. First, it is shown that if the plant is formulated modularly and the requirements specifications are expressed in a way standard to some engineering practices, and in addition follow some simple and easily checked rules called **CNMS** (laid out in detail in Section 3), then *the synthesis step is altogether unnecessary*: the plant and the requirement models already form a controllable, nonblocking, and maximally permissive supervisor. Knowing this beforehand is a huge benefit, as for such large-scale models state-of-the-art synthesis algorithms may take a long time, or may even not be able to synthesize a supervisor at all due to memory or time constraints.

However, the **CNMS** rules are rather conservative. Examples are known that do not fulfill the rules, but for which synthesis is still not necessary. Thus, the second contribution relaxes **CNMS** into **RCNMS**. Dependencies between the modular plant models based on the requirements are captured with a *dependency graph* (which does not need enumerating the actual state-space), where vertices relate to the plant models and the edges to the requirement models. By analyzing a dependency graph, it can be determined for **RCNMS** rules whether synthesis will be necessary or not. If there are no cycles, then the synthesis step is unnecessary: the plant and the requirement models already form a controllable, nonblocking, and maximally permissive supervisor. Note that, instead of using the more common suggestions found in the SCT literature to analyze the dependencies between plant models (e.g., shared events in Flordal and Malik (2009), Komenda et al. (2013)), the dependencies within the combined set of plant models and the requirement models is analyzed, as also suggested in Feng and Wonham (2006) and Goorden et al. (2020).

In general, the dependency graph has cyclic parts, though. The third contribution of this paper shows that synthesis can then be sectionalized, so as to be performed only for those plant and requirement models that are involved in the strongly connected components of those cycles. The other parts need no synthesis for the same reason as above. This results in modular supervisors, where a collection of supervisors controls the plant together. This contribution reduces the synthesis effort significantly, such that supervisors could now be synthesized for control problems where state-of-the-art synthesis algorithms fail (as demonstrated in Section 8).

The proposed method is most effective for plant models that are loosely coupled, often the result of using the input/output perspective (Balemi, 1992). Several case studies with real-life applications, such as Moormann et al. (2020) and Reijnen et al. (2018, 2017, 2020), have loosely coupled plant models, and experiments show that these models benefit greatly from the described approach. Even though some well-known examples (van der Sanden et al., 2015; Su et al., 2010; Wonham & Cai, 2019) are hard to fit in the presented framework, the key point is that *if* a system is modeled with the specific formulation described in this paper, *then* a reduction in synthesis effort can be achieved.

This paper builds upon preliminary results published in Goorden and Fabian (2019). While the **CNMS** model properties proposed in that paper capture the essence of some models of industrial applications, this paper generalizes the approach by providing relaxed conditions to determine that synthesis is not necessary.

Related work is Feng and Wonham (2006), where inspiration is taken from systems with shared resources, such as flexible manufacturing systems. In Feng and Wonham (2006), control-flow nets are introduced to analyze dependencies in the system and subsequently abstract away those parts of the system that will not contribute to a potential blocking issue. Control-flow nets are defined for shuffle systems with server and buffer specifications, which limits their applicability. In this paper, a notion similar to a shuffle system for the plant is used, while the specifications are in terms of state-based expressions, see Ma and Wonham (2005) and Markovski et al. (2010). Nevertheless, both works complement each other as they identify different classes of discrete-event systems for which synthesis is easy.

The structure of this paper is as follows. In Section 2, the preliminaries are provided. The **CNMS** properties as proposed in previous work (Goorden & Fabian, 2019) are presented in Section 3, and it is shown that for models satisfying these properties, synthesis is unnecessary. Section 4 introduces the dependency graph used to analyze the control problems. In Section 5, the result is established that synthesis is unnecessary when the dependency graph is acyclic. Section 6 extends the analysis to cyclic dependency graphs to reduce the original control problem into a set of smaller control problems. Sections 7 and 8 provide two case studies, related to a production line and to a roadway tunnel, to demonstrate the proposed analysis method. Section 9 concludes the paper.

2. Preliminaries

This section provides a brief summary of concepts related to automata, supervisory control theory, and directed graphs relevant for this paper. The concepts related to automata and supervisory control theory are taken from Cassandras and Lafortune (2008) and Wonham and Cai (2019). The concepts related to directed graphs are taken from Diestel (2017).

2.1. Automata

An automaton is a five-tuple $G = (Q, \Sigma, \delta, q_0, Q_m)$, where Q is the (finite) state set, Σ is the alphabet of events, $\delta : Q \times \Sigma \to Q$ the partial transition function, $q_0 \in Q$ the initial state, and $Q_m \subseteq Q$ the set of marked states. The alphabet $\Sigma = \Sigma_c \cup \Sigma_u$ is partitioned into two disjoint sets containing the controllable events (Σ_c) and the uncontrollable events (Σ_u), and Σ^* is the set of all finite strings of events in Σ, including empty string ϵ.

The notation $\delta(q,\sigma)!$ denotes that there exists a transition from state $q \in Q$ labeled with event σ, i.e., $\delta(q,\sigma)$ is defined. The transition function can be extended in a natural way to strings as $\delta(q,\epsilon) = q$ for the empty string ϵ, $\delta(q, s\sigma) = \delta(\delta(q,s),\sigma)$ where $s \in \Sigma^*$, $\sigma \in \Sigma$, and $\delta(q,s\sigma)!$ if $\delta(q,s)! \wedge \delta(\delta(q,s),\sigma)!$. The language generated by the automaton G is $\mathcal{L}(G) = \{s \in \Sigma^* \mid \delta(q_0,s)!\}$ and the language marked by the automaton G is $\mathcal{L}_m(G) = \{s \in \mathcal{L}(G) \mid \delta(q_0,s) \in Q_m\}$.

A path p of an automaton is defined as a sequence of alternating states and events, i.e., $q_1\sigma_1 q_2\sigma_2 \ldots \sigma_{n-1} q_n \sigma_n q_{n+1}$ s.t. $\forall i \in [1,n], \delta(q_i,\sigma_i) = q_{i+1}$. A path can also be written in the infix notation $q_1 \xrightarrow{\sigma_1} q_2 \xrightarrow{\sigma_2} \ldots \xrightarrow{\sigma_{n-1}} q_n \xrightarrow{\sigma_n} q_{n+1}$.

A state q of an automaton is called *reachable* if there is a string $s \in \Sigma^*$ with $\delta(q_0,s)!$ and $\delta(q_0,s) = q$. The automaton G is called *reachable* if every state $q \in Q$ is reachable. A state q is *coreachable* if there is a

string $s \in \Sigma^*$ with $\delta(q,s)!$ and $\delta(q,s) \in Q_m$. The automaton G is called *coreachable* if every state $q \in Q$ is coreachable. An automaton is called *nonblocking* if every reachable state is coreachable. An automaton is called *trim* if it is reachable and coreachable. Notice that a trim automaton is nonblocking, but a nonblocking automaton may not be trim, since it may have unreachable states.

An automaton is called *strongly connected* if from every state all other states can be reached, i.e., $\forall q_1, q_2 \in Q, \exists s \in \Sigma^*$ s.t. $\delta(q_1, s) = q_2$, see Ito (1978).

Two automata can be combined by synchronous composition.

Definition 1. Let $G_1 = (Q_1, \Sigma_1, \delta_1, q_{0,1}, Q_{m,1})$, $G_2 = (Q_2, \Sigma_2, \delta_2, q_{0,2}, Q_{m,2})$ be two automata. The synchronous composition of G_1 and G_2 is defined as

$$G_1 \parallel G_2 = (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta_{1\parallel 2}, (q_{0,1}, q_{0,2}), Q_{m,1} \times Q_{m,2})$$

where

$$\delta_{1\parallel 2}((x_1,x_2),\sigma) = \begin{cases} (\delta_1(x_1,\sigma), \delta_2(x_2,\sigma)) & \text{if } \sigma \in \Sigma_1 \cap \Sigma_2, \delta_1(x_1,\sigma)!, \text{ and } \delta_2(x_2,\sigma)! \\ (\delta_1(x_1,\sigma), x_2) & \text{if } \sigma \in \Sigma_1 \setminus \Sigma_2 \text{ and } \delta_1(x_1,\sigma)! \\ (x_1, \delta_2(x_2,\sigma)) & \text{if } \sigma \in \Sigma_2 \setminus \Sigma_1 \text{ and } \delta_2(x_2,\sigma)! \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Synchronous composition is associative and commutative up to reordering of the state components in the composed state set.

A composed system $\mathcal{G}$ is a collection of automata, i.e., $\mathcal{G} = \{G_1, \ldots, G_m\}$. The synchronous composition of a composed system $\mathcal{G}$, denoted by $\parallel \mathcal{G}$, is defined as $\parallel \mathcal{G} = G_1 \parallel \ldots \parallel G_m$, and the synchronous composition of two composed systems $\mathcal{G}_1 \parallel \mathcal{G}_2$ is defined as $\parallel (\mathcal{G}_1 \cup \mathcal{G}_2)$. A composed system $\mathcal{G} = \{G_1, \ldots, G_m\}$ is called a *product system* if the alphabets of the automata are pairwise disjoint, i.e., $\forall i, j \in [1,m], i \neq j, \Sigma_i \cap \Sigma_j = \emptyset$ (Ramadge & Wonham, 1989).

Finally, let G and K be two automata with the same alphabet Σ. K is said to be *controllable* with respect to G if, for every string $s \in \Sigma^*$ and $u \in \Sigma_u$ such that $\delta_K(q_{0,K}, s)!$ and $\delta_G(q_{0,G}, su)!$, it holds that $\delta_K(q_{0,K}, su)!$.

2.2. Supervisory control theory

The objective of supervisory control theory (Cassandras & Lafortune, 2008; Ramadge & Wonham, 1987, 1989; Wonham & Cai, 2019) is to design an automaton called a *supervisor* which function is to dynamically disable controllable events so that the closed-loop system of the plant and the supervisor obeys some specified behavior. More formally, given a plant model P and requirement model R, the goal is to synthesize supervisor S that adheres to the following control objectives.

- *Safety*: all possible behavior of the closed-loop system $P \parallel S$ should always satisfy the imposed requirements, i.e., $\mathcal{L}(P \parallel S) \subseteq \mathcal{L}(P \parallel R)$.
- *Controllability*: uncontrollable events may never be disabled by the supervisor, i.e., $P \parallel S$ is controllable with respect to P.
- *Nonblockingness*: the closed-loop system should be able to reach a marked state from every reachable state, i.e., $P \parallel S$ is nonblocking.
- *Maximal permissiveness*: the supervisor does not restrict more behavior than strictly necessary to enforce safety, controllability, and nonblockingness, i.e., for all other supervisors S' it holds that $\mathcal{L}(P \parallel S') \subseteq \mathcal{L}(P \parallel S)$.

Monolithic supervisory control synthesis results in a single supervisor S from a single plant model and a single requirement model (Ramadge & Wonham, 1987). There may exist multiple automata representations of the safe, controllable, nonblocking, and maximally permissive supervisor. Without loss of generality it is assumed that $S = P \parallel S$. When the plant model and the requirement model are given as a composed system $\mathcal{P}$ and $\mathcal{R}$, respectively, the monolithic plant model P and requirement model R are obtained by performing the synchronous composition of the models in the respective composed system.

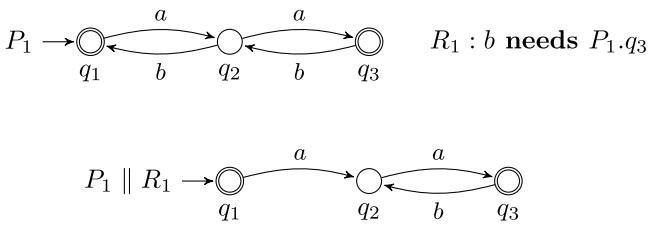

Fig. 1. An example of the synchronous composition of an automaton and a state-event invariant expression. In this and subsequent figures, (marked) locations are depicted with (concentric) circles, the initial location with an incoming arrow, and transitions with labeled edges.

For the purpose of supervisor synthesis, requirements can be modeled with automata and *state-based expressions* (Ma & Wonham, 2005; Markovski et al., 2010). The latter is useful in practice, as some control engineers tend to formulate requirements based on states of the plant. To refer to states of the plant, the notation $P.q$ is introduced that refers to state q of plant P. State references can be combined with the Boolean literals **T** and **F** and logic connectives to create *predicates*.

In this paper, state-event invariant expressions are considered. A *state-event invariant expression* formulates conditions on the enablement of an event based on states of the plant, i.e., the condition should evaluate to true for the event to be enabled. A state-event invariant expression is of the form σ **needs** C where σ is an event and C a predicate stating the condition. In general, event σ can be a controllable or an uncontrollable event. Let R be a state-event invariant expression, then $event(R)$ returns the event used in R and $cond(R)$ returns the condition predicate. The synchronous composition of a plant P with a state-event invariant expression R, denoted with $P \parallel R$, is defined by altering the transition function δ.

Definition 2. Let $P = (Q, \Sigma, \delta, q_0, Q_m)$ and $R = \mu$ **needs** C. Then the synchronous composition of P and R is defined as

$$P \parallel R = (Q, \Sigma, \delta', q_0, Q_m)$$

where $\delta'(q,\sigma) = \delta(q,\sigma)$, unless $\sigma = \mu$ and $C_{|P.q} = \mathbf{F}$, where $C_{|P.q}$ indicates that all state references $P.q$ in C are substituted by **T** and all state references $P.r, r \in Q, r \neq q$ in C replaced by **F**. In the latter case $\delta'(q,\sigma)$ is undefined.

An example to illustrate the synchronous composition between an automaton and a state-event invariant expression is provided in Fig. 1. This definition can be easily extended to a set of state-event invariant expressions $\mathcal{R} = \{R_1, \ldots, R_n\}$.

Given a composed system representation of the plant $\mathcal{P} = \{P_1, \ldots, P_m\}$ and a collection of requirements $\mathcal{R} = \{R_1, \ldots, R_n\}$, the tuple $(\mathcal{P}, \mathcal{R})$ is defined as the *control problem* for which a supervisor needs to be synthesized. The following assumptions are made about this control problem:

- $\mathcal{P} \neq \emptyset$, while $\mathcal{R}$ can be the empty set.
- For all $P \in \mathcal{P}$, it holds that P is an automaton where Q_P and Σ_P are nonempty.
- For all $R \in \mathcal{R}$, it holds that
  - if R is an automaton, then Q_R and Σ_R are nonempty, and $\Sigma_R \subseteq \Sigma_{\mathcal{P}}$ where $\Sigma_{\mathcal{P}} = \bigcup_{P \in \mathcal{P}} \Sigma_P$,
  - if R is a state-event invariant expression, then $event(R) \in \Sigma_{\mathcal{P}}$, and for each state reference $P.q$ in $cond(R)$ it holds that $P \in \mathcal{P}$ and $q \in Q_P$.

Modular supervisory control synthesis uses the fact that often the desired behavior is specified with a collection of requirements $\mathcal{R}$ (Wonham & Ramadge, 1988). Instead of first transforming the collection of

requirements into a single requirement, as monolithic synthesis does, modular synthesis calculates for each requirement a supervisor based on the plant model. In other words, given a control problem $(\mathcal{P}, \mathcal{R})$ with $\mathcal{R} = \{R_1, \ldots, R_n\}$, modular synthesis solves n control problems $(\mathcal{P}, \{R_1\}), \ldots, (\mathcal{P}, \{R_n\})$. Each control problem $(\mathcal{P}, \{R_i\})$ for $i \in [1, n]$ results in a safe, controllable, nonblocking, and maximally permissive supervisor S_i. The collection of supervisors $\mathcal{S} = \{S_1, \ldots, S_n\}$ can be conflicting, i.e., $S_1 \parallel \ldots \parallel S_n$ can be blocking. A nonconflicting check can verify whether $\mathcal{S}$ is nonconflicting (Mohajerani et al., 2016). In the case that $\mathcal{S}$ is nonconflicting, $\mathcal{S}$ is also safe, controllable, nonblocking, and maximally permissive for the original control problem $(\mathcal{P}, \mathcal{R})$. In the case that $\mathcal{S}$ is conflicting, an additional coordinator C can be synthesized such that $\mathcal{S} \cup \{C\}$ is safe, controllable, nonblocking, and maximally permissive for the original control problem $(\mathcal{P}, \mathcal{R})$ (Su et al., 2009).

2.3. Directed graphs

Definitions and notations of directed graphs are taken from Diestel (2017). A *directed graph* is a tuple (V, E) of sets of vertices V (or nodes) and edges E (or arcs), together with two functions init : $E \to V$ and ter : $E \to V$. The function init assigns to each edge e an initial vertex init(e) and the function ter assigns to each edge e a terminal vertex ter(e). An edge e is said to be directed from vertex init(e) to vertex ter(e). If init(e) = ter(e), the edge e is called a loop. A directed graph is called *self-loop free* if no edge is a loop. A directed graph $G' = (V', E')$ is a subgraph of $G = (V, E)$, denoted by $G' \subseteq G$, if $V' \subseteq V$ and $E' \subseteq E$.

A *path* in directed graph $G = (V, E)$ is a sequence of its vertices $p = x_0 x_1 \ldots x_k, k \geq 0$ such that for each step $i \in [0, k-1]$ there exists an edge $e_i \in E$ with init(e_i) = x_i and ter(e_i) = x_{i+1}. The path p is also called a path from x_0 to x_k. Two paths $p_1 = x_0 \ldots x_k$ and $p_2 = y_0 \ldots y_l$ can be concatenated into path $p_1 p_2 = x_0 \ldots x_k \ldots y_l$ if $x_k = y_0$. A *cycle* is a path $c = x_0 \ldots x_k x_0$ with $k \geq 1$, i.e., a cycle is a path from x_0 to itself with at least one other vertex along the path (a loop is not considered to be a cycle). A directed graph is called *cyclic* if it contains a cycle, otherwise it is called *acyclic*.

A directed graph is called *strongly connected* if there is a path between each pair of vertices. A *strongly connected component* of a directed graph is a maximal strongly connected subgraph.

3. Skipping synthesis for models with CNMS properties

This section describes first some characteristics of several applications where synthesis does not add any restrictions besides those implied by the requirements. Then, it provides properties that guarantee controllable, nonblocking, and maximally permissive supervisors that are together nonconflicting.

3.1. Characteristics of models

First, as the supervisors synthesized for the applications presented in Reijnen et al. (2018, 2017, 2020) are intended to be implemented on control hardware, the input–output perspective of Balemi (1992) is used. This entails that each sensor is modeled by uncontrollable events, while actuators are modeled by controllable events. Each event represents a change of the state of such a component. This modeling paradigm results in a collection of numerous small plant models that do not share any events. Therefore, the plant model is a product system.

In the rest of this paper, an automaton is called a *sensor automaton* if its alphabet has only uncontrollable events, i.e., $\Sigma = \Sigma_u$, and an *actuator automaton* if its alphabet has only controllable events, i.e., $\Sigma = \Sigma_c$.

Second, both sensors and actuators have cyclic behavior, often resulting in a trim and strongly connected plant model. For example, all sensors and actuators are modeled in this way in the production line in Reijnen et al. (2018). Furthermore, unreachable states in an uncontrolled plant represent states that are impossible to reach and are often not modeled or removed from the model.

Finally, requirements for applications often originate from safety risk analysis (Modarres, 2016) and failure mode and effect analysis (Stamatis, 2003). States are identified in which some actuator actions would result in unsafe behavior. For example, the safety specifications of a waterway lock that need to be fulfilled by the supervisor are mentioned in Section 4.191 of Rijkswaterstaat (2015). Each of the 16 requirements given over there describes a state of the system and the disablement of certain actuator actions for that state. It is shown in Reijnen et al. (2017) that these textual specifications can be described with state-event invariant expressions.

3.2. Properties

The following properties together guarantee that the control problem itself is a modular globally controllable and nonblocking system.

Definition 3 (***CNMS***)**.** A control problem $(\mathcal{P}, \mathcal{R})$ satisfies **CNMS** (Controllable and Nonblocking Modular Supervisors properties) if it has the following properties:

1. $\mathcal{P}$ is a product system.
2. For all $P \in \mathcal{P}$ holds that P is a strongly connected automaton with at least one marked state.
3. For all $R \in \mathcal{R}$ holds
    a. R is a state-event invariant expression μ **needs** C.
    b. $\mu \in \Sigma_c$.
    c. There exists no other requirement for this event μ.
    d. C is in a disjunctive normal form (see Davey and Priestley (1990)) where each atomic proposition (or variable) is of the form $P.q$ with $P \in \mathcal{P}$.
    e. Each conjunction contains at most one reference to each $P \in \mathcal{P}$.
    f. When $P \in \mathcal{P}$ only has a single state, the literal $\neg P.q$ is not allowed in C.
    g. Each $P \in \mathcal{P}$ mentioned in C is a sensor automaton.

The intuition behind the fact that a system satisfying **CNMS** is controllable and nonblocking is as follows. Properties 1 and 2 ensure that the plant is already nonblocking in the open loop setting, i.e. without controller, and exhibits cyclic behavior. Furthermore, they ensure that individual plant models behave independently of the other plant models, i.e. an individual plant model can take a transition while the state of each of the other plant models remains the same.

Requirements satisfying Property 3 will not introduce blocking or controllability issues. There is no controllability issue, as there may not exist a requirement restricting the enablement of uncontrollable events. The reason why the controlled system is still nonblocking can be explained as follows. First, a sensor automaton can always go to a marked state with Properties 1, 2 and 3.b. For a plant automaton with one or more controllable events, it is known from Properties 1 and 2 that from each state there exists a path to a marked state. For any controllable event along the path that is being restricted by a requirement, the condition of that requirement needs to be satisfied for the enablement of the controllable event. As only states of sensor automata are used in a condition and sensor automata can always reach each state without affecting other plant models, there exists a path in the sensor automata to satisfy the condition and subsequently enable the controllable event. By repeating the process of locally changing states in sensor automata, non-sensor automata can reach marked states if the requirements act as the supervisor.

The following theorem states that for a control problem satisfying **CNMS** synthesis can be skipped, i.e., the plant models and requirement models together already form controllable and nonblocking modular supervisors. In that case, the modular supervisor represented by the plant models and requirement models is by definition also maximally permissive. The proof of this theorem can be found in Appendix A.

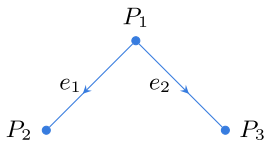

Fig. 2. The dependency graph G_{cp} of control problem $(\{P_1, P_2, P_3\}, \{R\})$ with $R = \mu$ **needs** $P_2.q_1 \vee \neg P_3.q_1$ and $\mu \in \Sigma_{P_1}$. This graph has three vertices P_1, P_2, and P_2 and two edges e_1 and e_2.

Theorem 1 (***CNMS*** *Goorden & Fabian, 2019*)**.** *Let $(\mathcal{P}, \mathcal{R})$ be a control problem satisfying **CNMS**. Then no supervisor synthesis is necessary, i.e., $\mathcal{P} \parallel \mathcal{R}$ is controllable and nonblocking, hence also maximally permissive.*

4. Dependency graphs of control problems

As indicated in Goorden and Fabian (2019), there exist published control problems that do not satisfy **CNMS**, but still do not require synthesis. In this section, the **CNMS** properties are relaxed.

4.1. Observations from models

The main reason the control problems of Reijnen et al. (2018, 2017, 2020) do not satisfy the **CNMS** properties is the violation of Property 3.g. In these control problems, there exist requirements that restrict the behavior of controllable events based on the behavior of plant models other than sensor automata, which in turn may also be restricted by other requirements. Several causes of this violation are described below.

As pointed out in Zaytoon and Carre-Meneatrier (2001), it may be desired to model the physical interaction between actuator and sensor components, because a supervisor that is proven to be deadlock-free for a model without interactions may deadlock after implementation on the physical system with interactions. Adding shared events to model the interactions will violate Property 1, as it is no longer a product system. Transforming this new model into a product system representation, the actuator and sensor models are combined into one due to the shared events. Therefore, requirements no longer refer only to states of sensor automata (violating Property 3.g).

Second, sometimes a requirement needs to refer explicitly to the state of an actuator to guarantee correct behavior of the system. For example, consider a hydraulic arm that has one actuator to extend it and one actuator to retract it. In this case, the modeler could express that it is undesired that both actuators are on at the same time, resulting in two requirements each expressing that one actuator may only be activated if the other actuator is deactivated.

Finally, timer-based requirements violate Property 3.g. A timer is typically modeled with a controllable event to activate it and an uncontrollable event to indicate the timeout of the timer. Therefore, the model of a timer is neither a sensor automaton nor an actuator automaton. If a timer is needed, typically two requirements associated with it express when it can be activated (the controllable events of the timer model are used) and what should happen when the timer has timed out (the state of the timer model is used). Service calls in a server–client architecture are modeled in the same way, see for example (Loose et al., 2018), where service calls are modeled with controllable events and responses with uncontrollable events.

4.2. Dependency graph

Control problems $(\mathcal{P}, \mathcal{R})$ satisfying all properties of **CNMS** *except* Property 3.g are called to satisfy the Relaxed Controllable and Nonblocking Modular Supervisors properties (**RCNMS**). For control problems satisfying **CNMS** or **RCNMS**, a directed graph can be constructed

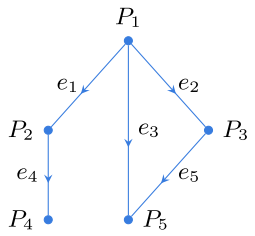

Fig. 3. A dependency graph G_{cp} of a control problem with five plant models satisfying **RCNMS** where P_4 and P_5 are sensor automata.

indicating the dependencies between plant models from $\mathcal{P}$ based on the requirement models from $\mathcal{R}$. In this directed graph, each vertex represents a plant model from the control problem. For each requirement in the control problem, a set of edges is present in the graph such that the initial vertex of each edge is the plant model containing the event that is restricted by the requirement. Furthermore, for each plant model used in the condition of the requirement there is an edge having this plant model as terminating vertex. For example, consider the control problem $(\{P_1, P_2, P_3\}, \{R\})$ with $R = \mu$ **needs** $P_2.q_1 \vee \neg P_3.q_1$ and $\mu \in \Sigma_{P_1}$. The dependency graph of this control problem is shown in Fig. 2. It has three vertices P_1, P_2 and P_3. For requirement R, two edges e_1 and e_2 are present such that $\text{init}(e_1) = \text{init}(e_2) = P_1$, as the restricted event of R originates from P_1, $\text{ter}(e_1) = P_2$, as P_2 is mentioned in the condition of R, and $\text{ter}(e_2) = P_3$, as P_3 is mentioned in the condition of R. This example also shows that there may be multiple, but isomorphic, dependency graphs for the same control problem.

More formally, let the *dependency graph* of control problem $(\mathcal{P}, \mathcal{R})$ be a directed graph $G_{cp} = (\mathcal{P}, E)$ s.t. $\forall R \in \mathcal{R}$ a set of edges $E_R \subseteq E$ is constructed s.t. $\forall e \in E_R$, $\text{init}(e) = P_i \in \mathcal{P} \wedge \mathit{event}(R) \in \Sigma_{P_i}$, and $\forall P_j \in \mathcal{P}$ used in $\mathit{cond}(R)$ $\exists e \in E_R$ with $\text{ter}(e) = P_j$, and finally $E = \bigcup_{R \in \mathcal{R}} E_R$.

A control problem satisfying **CNMS** results in an acyclic bipartite dependency graph.

5. Skipping synthesis with dependency graphs

Fig. 3 shows the dependency graph of a control problem satisfying **RCNMS**, but not **CNMS**. Plant models P_2 and P_3 have both incoming and outgoing edges, which indicate that the enablement of one or more events in each plant model is restricted by a requirement and that one or more states of each plant model are used in the condition of a requirement. Therefore, this model does not satisfy Properties 3.g of **CNMS**. This example demonstrates why the control problem underlying this graph still requires no synthesis.

For the **CNMS** property, it is shown with Theorem 1 that, essentially, no edge is permanently disabled. As the properties ensure that in a controlled system each sensor automaton can always reach each state, the condition of each state-event invariant expression can be eventually satisfied, enabling the controllable event of each state-event invariant expression. Therefore, each non-sensor plant model can reach all states from each state.

This argument can be used inductively to show that a control problem satisfying **RCNMS** still requires no synthesis. As the behavior of plants P_2 and P_3 in Fig. 3 only depends on sensor plants P_4 and P_5, it holds that P_2 and P_3 can reach all states from each state. Since the behavior of P_1 only depends on the plant models P_2, P_3, and P_5, and it is already known that all these models can reach all states from each state, it can conclude that P_1 also can reach all states from each state. Therefore, the complete control problem is controllable, nonblocking, and maximally permissive. This is formalized in Theorem 2. The proof of this theorem can be found in Appendix B. The result of this theorem is that if a control problem satisfies **RCNMS**, then no synthesis is needed.

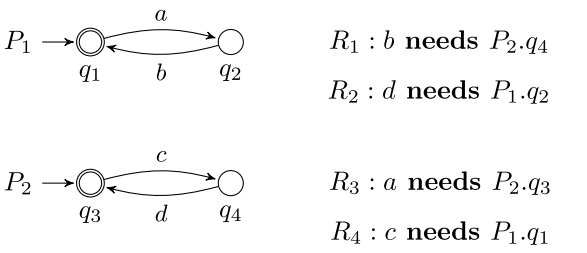

Fig. 4. Both control problems $CP_1 = (\{P_1, P_2\}, \{R_1, R_2\})$ and $CP_2 = (\{P_1, P_2\}, \{R_3, R_4\})$ result in cyclic dependency graphs, but the first contains a blocking issue while the second one does not.

Theorem 2 (*Acyclic **RCNMS***). *Let $(\mathcal{P}, \mathcal{R})$ be a control problem satisfying **RCNMS**. If the dependency graph G_{cp} of $(\mathcal{P}, \mathcal{R})$ is acyclic and self-loop free, then $\mathcal{P} \parallel \mathcal{R}$ is controllable and nonblocking, hence also maximally permissive.*

6. Sectionalizing control problems

For the case that a dependency graph is cyclic, supervisory control synthesis may be needed as $\mathcal{P} \parallel \mathcal{R}$ could be blocking. Fig. 4 shows two control problems $CP_1 = (\{P_1, P_2\}, \{R_1, R_2\})$ and $CP_2 = (\{P_1, P_2\}, \{R_3, R_4\})$, both based on the same set of plant models $\{P_1, P_2\}$. Those control problems result in the same cyclic dependency graph. However, CP_1 is blocking, while CP_2 is nonblocking.

Similarly, a dependency graph with a self-loop might also indicate that $\mathcal{P} \parallel \mathcal{R}$ is blocking. Consider again control problem CP_1, but now requirement R_2 is replaced by d **needs** $P_2.q_3$. This results in a self-loop in the dependency graph and the control problem is blocking. Yet, if requirement R_2 is replaced by d **needs** $P_2.q_4$, the dependency graph still has a self-loop, but the control problem is nonblocking.

So, a dependency graph containing cycles or self-loops may or may not require synthesis to obtain a controllable, nonblocking, and maximally permissive supervisor. In the remainder of this section it is shown that in case of a cyclic dependency graph the original control problem can be reduced to partial control problems containing the cycles.

6.1. Control problem reduction

From the dependency graph, all strongly connected components containing a cycle are identified. For each strongly connected component, the set of vertices (plant models) is denoted by ϕ, and the collection of these sets is denoted by $\Phi = \{\phi_1, \ldots, \phi_m\}$. From the definition of strongly connected components, it follows that they are non-overlapping. Fig. 5 shows control problem CP, with its dependency graph G_{CP} shown in Fig. 6. G_{CP} contains two cycles $c_1 = P_1P_2P_1$ and $c_2 = P_3P_4P_3$, and the strongly connected components of these two cycles are $\phi_1 = \{P_1, P_2\}$ and $\phi_2 = \{P_3, P_4\}$.

This example also shows plants whose behavior depends on the behavior of these strongly connected components. Requirement R_5 restricts the behavior of component model P_5 based on the behavior of component models P_2 and P_3. In this example, a supervisor is needed, as any synthesized supervisor for requirements R_1, R_2, R_3, and R_4 would make states $P_2.q_4$ and $P_3.q_6$ unreachable in the closed-loop system, and therefore requirement R_5 never enables event j. A supervisor is needed to disable event i to prevent component P_5 from being blocked in state q_{10}. Therefore, it is insufficient to only analyze the strongly connected components in isolation.

As a next step, vertices are added recursively to these strongly connected components. A vertex is added to a set of vertices if there exists an edge such that this edge originates in this added vertex and terminates in one of the vertices already in the set. Eventually, the strongly connected component is enlarged with those vertices from

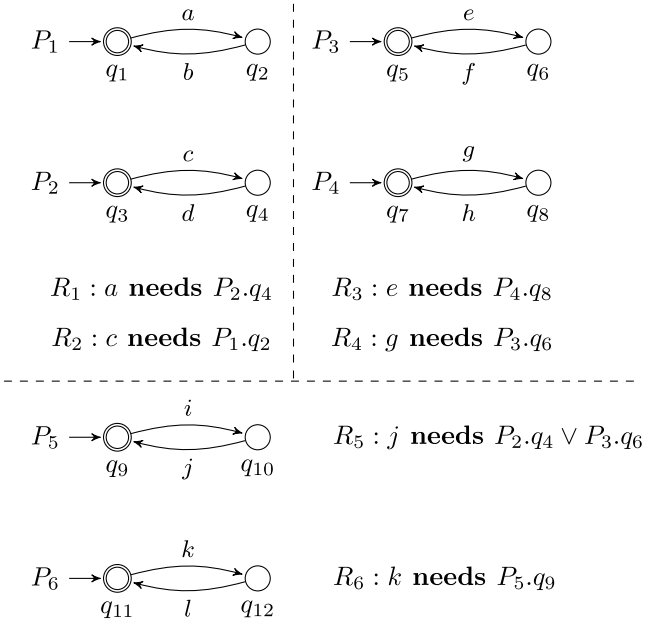

Fig. 5. A control problem $CP = (\{P_1, \ldots, P_6\}, \{R_1, \ldots, R_6\})$.

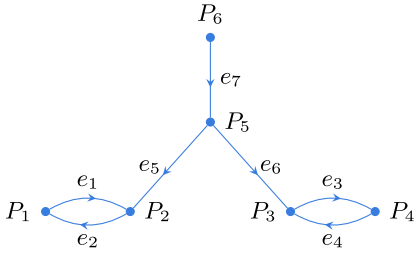

Fig. 6. The dependency graph of the control problem shown in Fig. 5.

which there exists a path to a vertex in the strongly connected component. Formally, the extended set of vertices for each strongly connected component with a cycle ϕ_i, denoted by V_{ϕ_i}, is defined as $V_{\phi_i} = \{P \in \mathcal{P} \mid \exists p = x_0x_1 \ldots x_k, k \geq 0, p \in Path(G_{CP})$ s.t. $x_0 = P \wedge x_k \in \phi_i\}$, and $\mathbb{V} = \{V_{\phi_1}, \ldots, V_{\phi_m}\}$, with $Path(G_{CP})$ the set of all paths in G_{CP}. The extended sets of vertices for the example are calculated as $V_{\phi_1} = \{P_1, P_2, P_5, P_6\}$ and $V_{\phi_2} = \{P_3, P_4, P_5, P_6\}$.

Still, it is insufficient to only analyze each extended vertex set V_{ϕ_i}. Two extended vertex sets may share vertices. This sharing could be problematic. In the running example, V_{ϕ_1} and V_{ϕ_2} share vertices P_5 and P_6.

Shared vertices between extended sets V_{ϕ_i} and V_{ϕ_j} will not always imply that it is necessary to analyze the partial control problem represented by $V_{\phi_i} \cup V_{\phi_j}$. Sometimes, it is still sufficient to analyze the partial control problems of V_{ϕ_i} and V_{ϕ_j} separately. For the control problem CP of Fig. 5, V_{ϕ_1} and V_{ϕ_2} should be combined, as the edges e_5 and e_6 relate to the same requirement R_5. The evaluation of the condition of requirement R_5 requires the result of the analysis of both strongly connected components ϕ_1 and ϕ_2. If requirement R_5 is replaced by, for example, the two requirements R'_5 : j **needs** $P_2.q_4$ and R''_5 : j **needs** $P_3.q6$, the extended sets V_{ϕ_i} and V_{ϕ_j} do *not* need to be merged for analyzing the cycles. While the dependency graph remains the same, edges e_5 and e_6 are now induced by different requirements.

Unfortunately, the above reasoning cannot be generalized. The control problem in Fig. 5 is modified again. An additional transition is added to plant model P_5 from state q_{10} to q_9 labeled with j'. Requirement R_5 is replaced by two requirements R'_5 : j **needs** $P_2.q_4$ and R''_5 : j' **needs** $P_3.q_6$. Again, the dependency graph in Fig. 6

remains unchanged. The controllable, nonblocking, and maximally permissive supervisor S_1 synthesized for the partial control problem $(\{P_1, P_2, P_5, P_6\}, \{R_1, R_2, R'_5, R_6\})$ would disable the transition labeled with event j, and the controllable, nonblocking, and maximally permissive supervisor S_2 synthesized for the partial control problem $(\{P_3, P_4, P_5, P_6\}, \{R_3, R_4, R''_5, R_6\})$ would disable the transition labeled with event j'. Now, $S_1 \parallel S_2$ is blocking, because plant P_5 deadlocks in state q_{10}, as the supervisors together disable both event j and event j'.

Therefore, two extended sets of vertices need to be merged once they share a vertex. Let $\sim \subseteq \mathbb{V} \times \mathbb{V}$ be a relation between extended sets of vertices. $(V_{\phi_i}, V_{\phi_j}) \in \sim$ if and only if $V_{\phi_i} \cap V_{\phi_j} \neq \emptyset$, i.e., they share at least one vertex, or $(V_{\phi_i}, V_{\phi_k}) \in \sim$ and $(V_{\phi_k}, V_{\phi_j}) \in \sim$ for some V_{ϕ_k}, i.e., they share a vertex with a common extended set of vertices. From this definition, it follows directly that $\sim$ is an equivalence relation, as it is reflexive, symmetric, and transitive.

Now, the partition $\mathbb{W}$ of $\mathbb{V}$ is the set of all equivalence classes of $\mathbb{V}$ with equivalence relation $\sim$, i.e., $\mathbb{W} = \mathbb{V}/\sim$ is the quotient set of $\mathbb{V}$ by $\sim$. For the example shown in Fig. 6, the partition $\mathbb{W}$ is $\{\{P_1, \ldots, P_6\}\}$.

A *simplified partial control problem* $(P'_s, \tilde{R}_s)$ *represented by a subset of vertices* $P'_s \subseteq \mathcal{P}$ is constructed as follows. First, $R'_s = \{R \in \mathcal{R} \mid \exists P \in P'_s \text{ s.t. } event(R) \in \Sigma_P\}$. Subsequently, the condition of each requirement in this set is adjusted where each literal containing reference to a state of a plant *not* in P'_s is replaced by the boolean literal **T**, resulting in the set of adjusted requirements $\tilde{R}_s$.

Theorem 3 contains the main result of this section: based on the dependency graph, synthesizing a supervisor can be performed following a modular approach which guarantees (global) controllability, nonblockingness, and maximal permissiveness. This theorem can be used to reduce the computational complexity of supervisor synthesis. The proof of this theorem can be found in Appendix C.

Theorem 3 (*Cyclic* ***RCNMS***). *Let $(\mathcal{P}, \mathcal{R})$ be a control problem satisfying* ***RCNMS*** *and let G_{cp} be its dependency graph. For each $W \in \mathbb{W}$, let S_W be a controllable, nonblocking, and maximally permissive supervisor for the simplified partial control problem represented by $\bigcup_{V \in W} V$. Then $\mathcal{P} \parallel \mathcal{R} \parallel (\|_{W \in \mathbb{W}} S_W)$ is a modular, controllable, nonblocking, and maximally permissive supervisor of $(\mathcal{P}, \mathcal{R})$.*

Theorem 3 shows for which partial control problems synthesis might still be needed and for which part of the system no synthesis is needed. In the worst-case situation, the original control problem is the only single equivalence class in $\mathbb{W}$ and no reduction can be achieved with the presented method. Sections 7 and 8 show that there exist industrial systems for which the control problem can be reduced. There are two options available for those partial control problems that might need synthesis: either synthesize a supervisor with an existing synthesis algorithm, like monolithic (Ramadge & Wonham, 1989), compositional (Mohajerani et al., 2014), and incremental synthesis (Brandin et al., 2004), or reason with an additional method that synthesis is still not needed (as it is known for the case studies in Reijnen et al. (2018, 2017, 2020) that no synthesis is needed). The second option is left open for future work.

7. Case study 1: FESTO production line

In this section, the proposed method is demonstrated with a case study. For this case study, a small-scale production line consisting of six workstations has been considered, see Fig. 7. The hardware of the system is produced by Festo Didactic for vocational training in the field of industrial automation. This system has been previously modeled in Reijnen et al. (2018). In the remainder of this section, first a description of this production line is provided. Subsequently, two workstations in isolation are analyzed to demonstrate Theorems 1 and 2. Finally, the complete production line is analyzed to demonstrate Theorem 3.

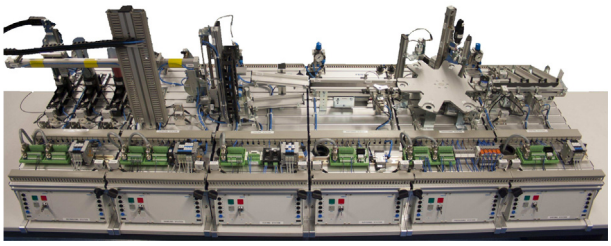

Fig. 7. Overview of the FESTO production line.

7.1. Case description

While no real production is taking place, all movements, velocities, and timings are as if it were. In total, the production line consists of 28 actuators, like DC motors and pneumatic cylinders, and 59 sensors, like capacitive, optical, and inductive ones.

The intended controlled behavior is as follows. Products enter the production line through the *distribution* station where they have been placed in three storage tubes. For each storage tube, a pusher is able to release a new product. The second workstation, the *handling* station, transports products from the distribution station to the testing station in two steps. First, a pneumatic gripper transports released products to an intermediate buffer. From this buffer, a transfer cylinder picks them up and places them in the *testing* station where the product height is tested. Correct products are moved by an air slide to the next station while rejected products are stored in a local buffer. In the fourth station, the *buffering* station, products can be held on a conveyor belt. A separator controls the release of products from the conveyor belt. At the next station, the *processing* station, products are processed. A turntable with six places rotates products through this station. After entering the processing station, the product is moved to a testing location where the orientation of the product is checked. Subsequently, at the next location a hole is drilled in the product only if the orientation is correct. At the fourth location, processed products are ejected to the sorting station. The last two locations can be used to correct the orientation if needed, and in that case the product can be processed again. In the final workstation, the *sorting* station, products are stored on one of the three slides, depending on color and the material of the product. Two pneumatic gates can be used to divert the product to the correct slide.

In Reijnen et al. (2018), a model of the production line is presented, which is slightly modified for this case study to have exclusively state-event invariant expressions; adjustments are indicated by comments in the model. The model contains 75 plant models and 214 requirement models, which can be accessed at a GitHub repository.[1]

Performing monolithic synthesis on this model reveals that the synthesized supervisor does not impose any additional restrictions to ensure controllable and nonblocking behavior, i.e., the control problem can already act as a modular, controllable, nonblocking, and maximally permissive supervisor.

7.2. Distribution station

The distributed construction of the model of the production line eases the individual analysis of workstations. To start with, the distribution station is analyzed.

Fig. 8 shows the dependency graph of the distribution station. To prevent cluttering of names, numbers are displayed in this and subsequent figures instead of the actual plant names in the model. The readme file in the model repository explains how the actual names can be obtained. Plant models 1 through 10 are sensor automata, i.e., they only have uncontrollable events in their alphabet, plant models 11, 12,

[1] https://github.com/magoorden/NonblockingModularSupervisors.

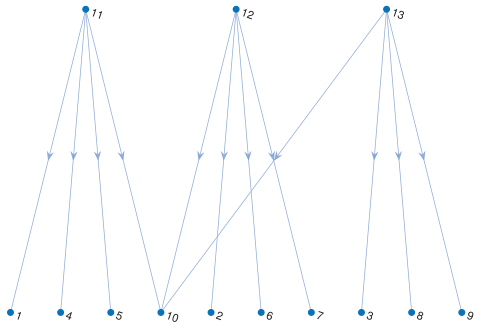

Fig. 8. The dependency graph of the distribution station. For readability, numbers are used as nodes instead of the actual names from the model.

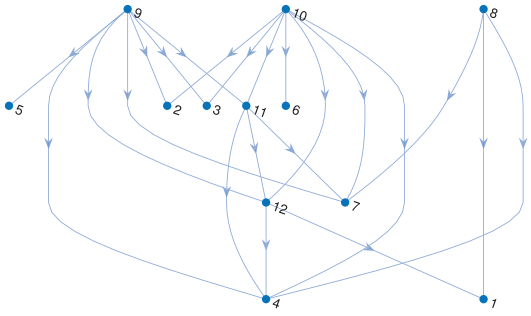

Fig. 9. The dependency graph of the sorting station.

and 13 are actuator automata, i.e., they only have controllable events in their alphabet. As each edge in this dependency graph has an actuator automaton as initial vertex and a sensor automaton as terminal vertex, Theorem 1 applies. This indicates that, if a supervisor is only needed for this workstation, synthesis can be skipped and the control problem already represents the supervisor.

7.3. Sorting station

Fig. 9 shows the dependency graph of the sorting station. In this workstation, plant models 1 through 7 are sensor automata, plant models 8 through 11 are actuator automata and plant model 12 contains both controllable and uncontrollable events. This graph already indicates that Theorem 1 does not apply: there are edges (representing requirements) that have a non-sensor automaton as a terminal vertex. In particular, plant models 11 and 12 have both incoming and outgoing edges, which indicates a violation of Property 3.g of the **CNMS** properties. Fortunately, as the model satisfies the **RCNMS** properties and the control dependency graph is acyclic, Theorem 2 applies. Therefore, synthesis can be skipped.

7.4. Production line

Fig. 10 shows the dependency graph of the complete production line. Cycles in this graph are indicated in red. Clearly, both Theorems 1 and 2 are not applicable for the control problem of the complete production line.

With the help of Theorem 3, the problem of synthesizing a monolithic supervisor can be reduced to analyzing smaller control problems based on the identified cycles. In the dependency graph, five strongly connected components containing cycles can be identified: $\phi_1 = \{P_{21}, P_{22}\}$, $\phi_2 = \{P_{25}, P_{26}\}$, $\phi_3 = \{P_{36}, P_{37}\}$, $\phi_4 = \{P_{47}, P_{48}\}$, and $\phi_5 = \{P_{58}, P_{59}, P_{60}, P_{61}, P_{62}\}$. Next, these sets need to be extended to include all plant models from which there exists a path to one of the plants in that particular strongly connected component. This is only the case for ϕ_1, as from P_{23} there exists a path from P_{23} to P_{21} (and P_{22}).

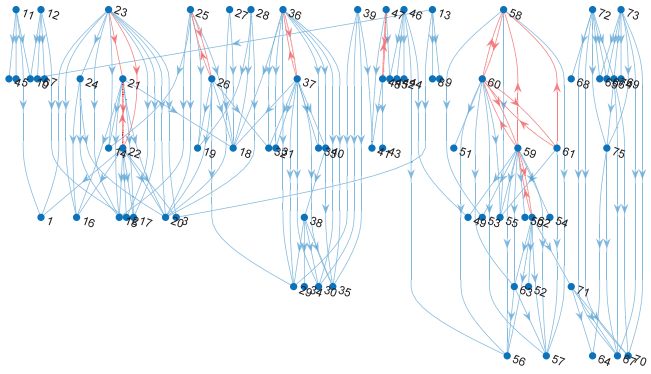

Fig. 10. The dependency graph of the complete production line. Red color indicates cycles. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Therefore, $V_{\phi_1} = \{P_{21}, P_{22}, P_{23}\}$, while $V_{\phi_2} = \phi_2$, $V_{\phi_3} = \phi_3$, $V_{\phi_4} = \phi_4$, and $V_{\phi_5} = \phi_5$. In this case, there is no overlap between these extended sets, so $W_i = V_{\phi_i}$ for $i \in [1, 5]$.

Finally, five supervisors, $S_1, \ldots, S_5$ are synthesized, one for each simplified partial control problem represented by $\bigcup_{V_{\phi_i} \in W} V_{\phi_i}$. From Theorem 3 it follows that $\mathcal{P} \parallel \mathcal{R} \parallel S_1 \parallel S_2 \parallel S_3 \parallel S_4 \parallel S_5$ is a modular, controllable, nonblocking, and maximally permissive supervisor for the production line.

Table 1 shows the results of applying Theorem 3 on the production line model. For each control problem solved, the uncontrolled and controlled state-space size is provided. The control problems for synthesizing automaton-based supervisors $S_1, \ldots, S_5$ are tiny compared to monolithic synthesis, i.e., obtaining these supervisors can be done even manually. In future research, a full experimental analysis of potential computational effort reduction with respect to other synthesis algorithms can be performed. Inspecting the synthesized supervisors confirms the observation from Section 7.1 that no additional restrictions are imposed to ensure controllable and nonblocking behavior.

8. Case study 2: Roadway tunnel

In this section, the applicability of the proposed method is demonstrated on an industrial large-scale system. For this demonstration, the case study of synthesizing a supervisory controller for the 'Eerste Heinenoord Tunnel', a tunnel located south of Rotterdam, the Netherlands, is used. The model of this system is described in Moormann et al. (2020).

8.1. Case description

Nowadays, each tunnel is equipped with a supervisory controller that ensures correct cooperation between the tunnel subsystems, such as ventilation, lighting, boom barriers, and emergency detection sensors. For example, when an emergency is detected by several sensors, the supervisor has to automatically close off the tunnel for traffic. Fig. 11 shows the 'Eerste Heinenoord Tunnel' (EHT) on the right and the 'Tweede Heinenoord Tunnel' (THT) on the left. The EHT is a two-tube roadway tunnel, which was initially opened in 1969. The THT, which was added in 1999, is only accessible for slow traffic such as cyclists and agricultural traffic. Rijkswaterstaat, the executive body of the Dutch ministry of Infrastructure and Water Management, is currently in the preparation and planning phase of renovating the EHT. In the renovation project, both the physical tunnel components and the tunnel supervisory controller are being renewed.

The model of the EHT in Moormann et al. (2020) contains 540 plant models and 1668 requirement models, which can be accessed at a

Table 1
Results of supervisory control synthesis for the production line.

Model	Uncontrolled state-space size	Controlled state-space size	Synthesis duration [s]
Monolithic	$5.9 \cdot 10^{26}$	$2.2 \cdot 10^{25}$	370
S_1	8	6	<1
S_2	4	3	<1
S_3	4	3	<1
S_4	6	6	<1
S_5	512	76	<1

Fig. 11. The Eerste Heinenoord Tunnel (right) and the Tweede Heinenoord Tunnel (left). Image from https://beeldbank.rws.nl, Rijkswaterstaat.

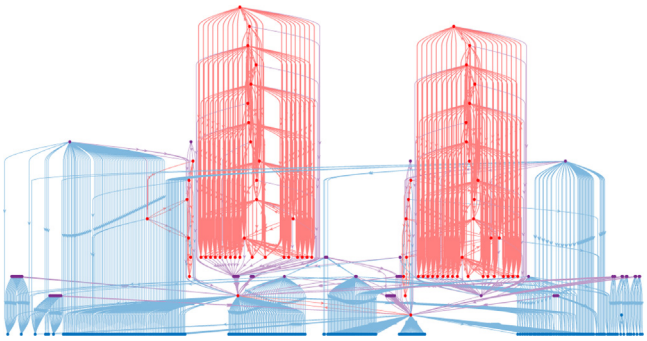

Fig. 12. The dependency graph of the EHT. Red color indicates the five strongly connected components, purple color the nodes and edges added in the extended strongly connected components, and blue color the nodes and edges that can be omitted from synthesis according to Theorem 3. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

GitHub repository.[2] This large number of component models results in the uncontrolled state-space size of $1.87 \cdot 10^{226}$, for which a monolithic supervisor can no longer be calculated by the CIF tooling (van Beek et al., 2014).

8.2. Results

The model of the EHT satisfies **RCNMS**, but it does not satisfy **CNMS**. Therefore, Theorem 1 does not apply. Fig. 12 shows the dependency graph of this model. Again, extended cycles are indicated in red in the figure. Since the dependency graph is cyclic, Theorem 2 does not apply too. Therefore, Theorem 3 is used to reduce the synthesis problem.

With the help of Theorem 3, instead of using the complete model as input for synthesis, the model can be significantly reduced. The dependency graph of the EHT model contains five strongly connected components, which transforms into one large subgraph of the five extended sets of vertices. Now, according to Theorem 3, all blue vertices (and edges) can be removed before synthesis is started on the control problem represented by the red and purple edges and vertices.

Table 2 shows the results of the analysis of the EHT. In the most-refined product representation, the EHT model contains 492 plant models and 1668 requirements. Theorem 3 reduces the synthesis problem to only 157 plant models and 1312 requirement models. This is a reduction of 68% of the plant models and 21% of the requirement models. Now the reduced model can be used as input for any synthesis method, e.g. monolithic, modular, and compositional synthesis, to obtain a supervisor. Monolithic synthesis is applied to verify whether a supervisor can be synthesized for the reduced control problem without running into memory issues. For the reduced control problem, a monolithic supervisor can be synthesized in 19.4 s. This shows that reducing the control problem is beneficial for synthesis.

As a subsequent experiment, multilevel synthesis (Goorden et al., 2020; Komenda et al., 2016) and compositional synthesis (Mohajerani et al., 2014) are applied on the original model of the EHT. For multilevel synthesis, the implementation in CIF (van Beek et al., 2014) is used; for compositional synthesis, the implementation in Supremica (Malik et al., 2017) is used. Multilevel synthesis is able to synthesize supervisors on average in 220 seconds.[3] This is without performing a nonconflicting check on the synthesized supervisors. Both the monolithic BDD-based nonconflicting check in CIF and the compositional nonconflicting check in Supremica run out of memory (4 GB available). Compositional synthesis is not able to synthesize a supervisor, because it runs out of memory (4 GB available). This experiment shows that it is currently sometimes necessary to reduce the control problem before performing state-of-the-art synthesis algorithms on models of large-scale applications.

9. Conclusion

This paper presents contributions to determine, based on model properties, whether synthesis is unnecessary for a given set of modular plant models and requirement models, building upon preliminary results presented in the conference paper of Goorden and Fabian (2019). These contributions result in the following effective three-step method for synthesizing supervisors. First, it is checked whether a control problem satisfies the **CNMS** properties. If it does, then the synthesis step is altogether unnecessary: the plant and the requirement models already form a controllable, nonblocking, and maximally permissive supervisor. If not, the second step is to check whether the control problem satisfies the relaxed **RCNMS** properties and to construct its dependency graph, where vertices relate to the plant models and the edges to the requirement models. If the dependency graph is acyclic, then the synthesis step is still unnecessary. If it has cycles, the third step is to reduce the original control problem to a collection of smaller partial control problems, one for each strongly connected component in the dependency graph. This results in modular supervisors which control the plant together.

[2] https://github.com/magoorden/NonblockingModularSupervisors.

[3] With clustering settings of $\alpha = 2, \beta = 5.0$, and $\mu = 2.0$.

Table 2
Results of supervisory control synthesis for the EHT. Monolithic synthesis has been used.

Model	Original control problem	Reduced control problem
Number of plant models	492	157
Number of requirement models	1668	1312
Uncontrolled state-space size	$1.87 \cdot 10^{226}$	$1.48 \cdot 10^{87}$
Controlled state-space size	–	$2.55 \cdot 10^{81}$
Synthesis duration [s]	–	19.4

Two industrial cases studies demonstrate that the method presented in this paper generates useful results in practice by significantly reducing the synthesis effort. The tunnel case study even shows that model reduction is necessary, because state-of-the-art synthesis tools are not able to provide supervisors for the original model.

The infrastructural systems encountered in the project with Rijkswaterstaat, like waterway locks (Reijnen et al., 2017), movable bridges (Reijnen et al., 2020), and tunnels (Moormann et al., 2020), satisfy **RCNMS**. This is a motivation to further investigate the applicability of the proposed model properties and analysis method to systems from other domains, like manufacturing and automotive systems.

Future work also includes the identification of special cases to be able to conclude that the synthesis step is unnecessary for some of the partial control problems identified by the strongly connected components. Monolithic supervisors of the partial control problems of the production line case still indicate that the synthesis step is unnecessary, but it is yet unclear how this conclusion could be obtained without having performed synthesis. Another direction is to investigate the applicability of the presented method if the supervisors are not required to be maximally permissive, i.e., if the goal is to synthesize controllable, and nonblocking supervisors that achieve something but not necessarily everything possible.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The authors thank Ferdie Reijnen for providing the models of the Festo production line and Lars Moormann for providing the models of the Eerste Heijnenoord Tunnel. Advice given by prof.em. Jan H. van Schuppen has been a great help in preparing the manuscript of this paper. The authors thank Han Vogel and Maria Angenent from Rijkswaterstaat for their valuable feedback and support.

This work is supported by Rijkswaterstaat, the Netherlands, part of the Ministry of Infrastructure and Water Management of the Government of the Netherlands, and by the Swedish Science Foundation, Vetenskapsrådet.

Appendix A. Proof of Theorem 1

The proof of Theorem 1 as presented in this section originates from the conference proceedings (Goorden & Fabian, 2019). In order to prove that a control problem satisfying **CNMS** does not require synthesis (Theorem 1), the following five lemmas are first proved.

The first two lemmas show that when a plant model is provided as a product system and each individual automaton is trim or strongly connected, then the synchronous composition of these automata is also trim or strongly connected, respectively.

Lemma 1. *Let $\mathcal{P} = \{P_1, \ldots, P_m\}$ be a product system where each individual $P_i \in \mathcal{P}$ is trim. Then $P_1 \parallel \ldots \parallel P_m$ is trim.*

Proof. Denote $P = P_1 \parallel \ldots \parallel P_n$, $P = (Q, \Sigma, \delta, q_0, Q_m)$, and $P_i = (Q_i, \Sigma_i, \delta_i, q_{0,i}, Q_{m,i})$. It is shown that P is reachable and coreachable.

Firstly, assume that $q = (q_1, \ldots, q_n)$ is a state in P. As each individual P_i is trim, it follows that there exists a string $s_i \in \Sigma_i^*$ such that $\delta_i(q_{0,i}, s_i) = q_i$. From the definition of synchronous composition and the fact that $\mathcal{P}$ is a product system, it follows that $\delta((r_1, \ldots, q_{0,i}, \ldots, r_m), s_i) = (r_1, \ldots, q_i, \ldots, r_m)$ for any state $r_j \in Q_j, j \neq i$. Therefore, it holds that $\delta((q_{0,1}, \ldots, q_{0,n}), s_1 s_2 \ldots s_n) = q$ in P. As the state q is chosen arbitrarily, it follows that P is reachable.

Secondly, assume again that $q = (q_1, \ldots, q_n)$ is a state in P. As each individual P_i is trim, it follows that there exists a string $s_i \in \Sigma_i^*$ such that $\delta_i(q_i, s_i) = q_{i,k} \in Q_{m,i}$. From the definition of synchronous composition and the fact that $\mathcal{P}$ is a product system, it follows that $\delta((r_1, \ldots, q_i, \ldots, r_m), s_i) = (r_1, \ldots, q_{i,k}, \ldots, r_m)$ for any state $r_j \in Q_j, j \neq i$. Therefore, it holds that $\delta(q, s_1 s_2 \ldots s_n) \in Q_m$ in P. As state q is chosen arbitrarily, it follows that P is coreachable. □

Lemma 2. *Let $\mathcal{P} = \{P_1, \ldots, P_m\}$ be a product system where each individual $P_i \in \mathcal{P}$ is a strongly connected automaton. Then $P_1 \parallel \ldots \parallel P_m$ is a strongly connected automaton.*

Proof. Denote $P = P_1 \parallel \ldots \parallel P_n$, $P = (Q, \Sigma, \delta, q_0, Q_m)$, and $P_i = (Q_i, \Sigma_i, \delta_i, q_{0,i}, Q_{m,i})$. It is shown that for any two states $x = (x_1, \ldots, x_m) \in Q, y = (y_1, \ldots, y_m) \in Q$ there exists a string $s \in \Sigma^*$ such that $\delta(x, s) = y$.

As each individual P_i is strongly connected, it follows that there exists a string $s_i \in \Sigma_i^*$ such that $\delta_i(x_i, s_i) = y_i$. From the definition of synchronous composition and the fact that $\mathcal{P}$ is a product system, it follows that $\delta((r_1, \ldots, x_i, \ldots, r_m), s_i) = (r_1, \ldots, y_i, \ldots, r_m)$ for any state $r_j \in Q_j, j \neq i$. Therefore, it holds that $\delta(x, s_1 s_2 \ldots s_n) = y$ in P. As states x and y are chosen arbitrarily, it follows that P is a strongly connected automaton. □

The following lemma expresses that when a control problem with a single requirement satisfies **CNMS**, then always eventually a state can be reached such that the condition of this requirement evaluates to true, thus enabling the guarded event.

Lemma 3. *Let $(\mathcal{P}, \{R\})$ be a control problem with a single requirement satisfying **CNMS**. Denote $R = e$ **needs** C. Then, from any state q, there exists a string $s \in \Sigma^*$ such that a state r is reached and $C(r) = \mathbf{T}$.*

Proof. As $\mathcal{P}$ is a product system (Property 1), there is only a single plant component P_k such that $e \in \Sigma_k$. From the combination of Properties 3.b, 3.d, and 3.g, it follows that plant component P_k is not used in condition C, as it has to be an actuator model. Therefore, the state of P_k does not matter.

Furthermore, observe that $\mathcal{P} \setminus \{P_k\} \neq \emptyset$ and $\parallel (\mathcal{P} \setminus \{P_k\}) = \parallel (\mathcal{P} \setminus \{P_k\}) \parallel R$. From Property 2 and Lemma 2 it follows that $\parallel (\mathcal{P} \setminus \{P_k\})$ is a strongly connected automaton, thus $\parallel (\mathcal{P} \setminus \{P_k\}) \parallel R$ is also a strongly connected automaton. Therefore, if there exists a state r that satisfies C, i.e., $C(r) = \mathbf{T}$, then there also exists a string $s \in \Sigma^*$ such that $\delta(q, s) = r$. So it remains to be proven that such a state r exists.

As C is in disjunctive normal form (Property 3.d), it follows that if r satisfies C, it satisfies one of the conjunctions. From Properties 3.e and 3.g it is known that there is at most one reference to each $P_i \in \mathcal{P} \setminus \{P_k\}$ in each conjunction. If there is no reference to P_i, then all states of this automaton satisfy this conjunction. If P_i is mentioned in this conjunction, then, from Properties 3.d and 3.f, there exists at least one state $q_i \in Q_i$ that satisfies this conjunction. Thus there exists a state r such that C is satisfied. □

Now the following two lemmas are proven: the first one shows that under the given conditions, synthesis is not needed to be performed locally, and the second one shows that under the given conditions the supervisors are globally nonblocking. In the rest of this section, the notation $\sup CN(P,R)$ is the function that constructs the controllable, nonblocking, and maximally permissive supervisor given plant P and requirement R.

Lemma 4. *Let $(\mathcal{P},\mathcal{R})$ be a control problem satisfying **CNMS**. For each $R_j \in \mathcal{R}$, $P \parallel R_j$ is a controllable, nonblocking, and maximally permissive supervisor for plant $P = \parallel \mathcal{P}$ and requirement R_j.*

Proof. In the case that $\mathcal{R} = \emptyset$, no supervisor is synthesized. It follows from Properties 1 and 2 and Lemma 1 that P is trim, so there is indeed no need for a supervisor. In the remainder of the proof it is assumed that $\mathcal{R} \neq \emptyset$.

For each individual supervisor $P \parallel R_j$ it is shown below that $P \parallel R_j$ is controllable with respect to plant P and that $P \parallel R_j$ is nonblocking. The fact that $P \parallel R_j$ is controllable follows directly from Property 3.b. It remains to be proven that $P \parallel R_j$ is nonblocking. From Property 3.a it follows that an event $e_j = event(R_j)$ is associated with this requirement R_j. As $\mathcal{P}$ is a product system (Property 1), there is only a single plant component P_k such that $e_j \in \Sigma_k$. Now the set of plant component models is partitioned into $\{P_k\}$, $P_{sm} = \{P_i \in \mathcal{P} \mid P_i \text{ is a sensor model}\}$, and $P_o = \mathcal{P} \setminus (\{P_k\} \cup P_{sm})$. Observe that the behavior of the plant components in P_{sm} and P_o is not restricted by requirement R_j, so Lemmas 1 and 2 apply to the sets P_{sm}, P_o, and $P_{sm} \cup P_o$, i.e, $P_{sm} \parallel R_j$, $P_o \parallel R_j$, and $(P_{sm} \cup P_o) \parallel R_j$ are all trim and strongly connected automata.

To show that $P \parallel R_j$ is nonblocking, it is shown next that for each reachable state q there exists a string $s \in \Sigma^*$ such that a marked state $q_m \in Q_m$ can be reached. Consider automaton P_k with current state q_k. As automaton P_k is trim (Property 2), there exists a path labeled with string $s_k \in \Sigma_k^*$ by which a state $q_{m,k} \in Q_{m,k}$ can be reached from state q_k. It is shown that this path is still possible under the influence of requirement R_j, i.e., it is still a path in $P_k \parallel R_j$. Consider two cases for this path.

- If s_k does not contain event e_j, then the path labeled with s_k is trivially possible in $P_k \parallel R_j$.
- If s_k contains event e_j, then requirement R_j may remove event e_j from the enabled event set and prevents $P_k \parallel R_j$ from reaching a marked state. For each transition labeled with event e_j, Lemma 3 expresses that there exists a path in P reaching a state r such that $C(r) = \mathbf{T}$. Therefore, there always exists a path in P such that e_j is enabled. Thus, the path labeled with s_k is still possible in $P_k \parallel R_j$.

Combining the above observation for s_k and the fact that $(P_{sm} \cup P_o) \parallel R_j$ is trim, it follows that a string s exists by which a marked state q_m is reached from state q. As q is arbitrarily chosen, it follows that $P \parallel R_j$ is nonblocking. □

Lemma 5. *Let $(\mathcal{P},\mathcal{R})$ be a control problem satisfying **CNMS**. Construct the set of modular supervisors $\mathcal{S} = \{S_1,\ldots,S_n\}$ such that each supervisor $S_j = \sup CN(P,R_j)$ is the controllable, nonblocking, and maximally permissive supervisor for plant $P = P_1 \parallel \ldots \parallel P_m$ and requirement $R_j \in R$. Then $\mathcal{S}$ is nonconflicting.*

Proof. For $\mathcal{S}$ to be nonconflicting, it should hold that $S_1 \parallel \ldots \parallel S_n$ is nonblocking. From Lemma 4 it follows that each $S_j = P \parallel R_j$. Therefore, $S_1 \parallel \ldots \parallel S_n = (P \parallel R_1) \parallel \ldots \parallel (P \parallel R_n) = P \parallel R_1 \parallel \ldots \parallel R_n$. Partition the set of plant models $\mathcal{P}$ into the set of sensor models $P_{sm} = \{P_i \in \mathcal{P} \mid P_i \text{ is a sensor model}\}$, the set of restricted models $P_r = \{P_i \in \mathcal{P} \mid \exists R_j \in \mathcal{R} \text{ s.t. } event(R_j) \in \Sigma_i\}$, and the other plant models $P_o = \mathcal{P} \setminus (P_{sm} \cup P_r)$.

Clearly, no plant model in P_o is affected by the requirements, so Lemmas 1 and 2 apply, i.e., $P_o \parallel \mathcal{R}$ is a trim and strongly connected automaton. Furthermore, from Property 3.b and the definition of a sensor model it follows that also no plant model in P_{sm} is affected by the requirements, thus by Lemmas 1 and 2 it follows that $P_{sm} \parallel \mathcal{R}$ is a trim and strongly connected automaton. Again using Lemmas 1 and 2 yields that $P_o \parallel P_{sm} \parallel \mathcal{R}$ is a trim and strongly connected automaton.

For $P_o \parallel P_{sm} \parallel P_r \parallel \mathcal{R}$ to be nonblocking, it should hold that from every reachable state $q \in Q$ there exists a string $s \in \Sigma^*$ such that $\delta(q,s) \in Q_m$. As P_r is trim (Lemma 1) it follows that there exists a string $s_r \in \Sigma_r^*$ such that $\delta(q_r,s_r) \in Q_m$ in P_r. For $\delta(q_r,s_r) \in Q_m$ in $P_r \parallel \mathcal{R}$ to exist, each event in s_r should be enabled along its path. There are two cases for each event σ in string s_r to consider following Definition 2 of synchronous composition with a state-event requirement.

- If there does not exist a requirement $R_j \in \mathcal{R}$ such that $event(R_j) = \sigma$, then σ is enabled.
- If there does exist a requirement $R_j \in \mathcal{R}$ such that $event(R_j) = \sigma$, then R_j is also the only requirement in $\mathcal{R}$ such that $event(R_j) = \sigma$ (Property 3.c). As the condition $C_j = cond(R_j)$ only depends on plant components from P_{sm} and not plant components from P_r or P_o (Property 3.g), it follows from Lemma 4 that there exists a string in P_{sm} such that the reached state r satisfies C_j. No transition in plant components from P_r and P_o are needed as all states from these plant components are irrelevant in satisfying the condition C_j. Therefore, there exists a path in P such that σ is enabled.

From the above observation, it can be concluded that always a string (including the empty string) such that σ is enabled can be found. As σ is chosen arbitrarily along the path in P_r labeled with s_r, it follows that $\delta(q_r,s_r) \in Q_{m,r}$. Finally, combining this with the fact that q_r is chosen arbitrarily and that $P_o \parallel P_{sm} \parallel \mathcal{R}$ is trim, it follows that $P_o \parallel P_{sm} \parallel P_r \parallel \mathcal{R}$ is nonblocking. □

Now Theorem 1 is proven.

Proof of Theorem 1. From Lemmas 4 and 5 it follows that a set of supervisors $\mathcal{S} = \{S_1,\ldots,S_n\}$ can be constructed such that $S_j = \sup CN(P,R_j) = P \parallel R_j$ and $\mathcal{S}$ is nonconflicting. The antecedent follows directly from combining these last two facts. □

Appendix B. Proof of Theorem 2

Before Theorem 2 is proven, the following lemma is introduced which transforms an acyclic dependency graph into a forest of trees. A *tree* is an acyclic directed graph where each vertex has at most one incoming edge, i.e., for each vertex v there is at most one edge e such that $\text{ter}(e) = v$. A forest is a set of trees. A forest can be constructed from an acyclic directed graph recursively. Assume that a subgraph T having vertex v as root node is already a tree. Then for each incoming edge into v subgraph T is duplicated and set to the terminating vertex of that edge. Fig. B.13 shows the forest with a single tree of the dependency graph as shown in Fig. 3. As vertex P_5 has two incoming edges, the directed graph G_{cp} is not a tree. By duplicating vertex P_5, the tree in Fig. B.13 is constructed.

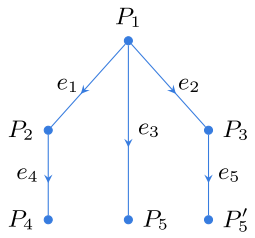

Fig. B.13. The forest of dependency graph G_{cp} from Fig. 3.

From a dependency (sub)graph, the control problem it represents can be reconstructed as follows. The control problem $(\mathcal{P}, \mathcal{R}')$ represented by a dependency graph $(\mathcal{P}, E)$ is the one where $\mathcal{R}' = \{R \in \mathcal{R} \mid \exists e \in E \text{ s.t. } event(R) \in \Sigma_{\text{init}(e)} \wedge \text{ter}(e) \in cond(R)\}$.

Lemma 6. *Let $G_{CP} = (\mathcal{P}, E)$ be an acyclic dependency graph of control problem $CP = (\mathcal{P}, \mathcal{R})$ satisfying **RCNMS**, and let F be the forest constructed from G_{CP}. Then S is a controllable, nonblocking, and maximally permissive supervisor of CP if and only if S is a controllable, nonblocking, and maximally permissive supervisor of the control problem represented by F.*

Proof. In the construction of the forest F from G_{CP}, subgraphs are duplicated. Duplicating plants and requirements results in the same controllable, nonblocking, and maximally permissive supervisor, i.e., S is a controllable, nonblocking, and maximally permissive supervisor for $(\mathcal{P}' \parallel \mathcal{P}') \parallel (\mathcal{R}' \parallel \mathcal{R}')$ if and only if S is a controllable, nonblocking, and maximally permissive supervisor for $\mathcal{P}' \parallel \mathcal{R}'$, where $\mathcal{P}' \subseteq \mathcal{P}$ and $\mathcal{R}' \subseteq \mathcal{R}$ are sets of plant models and requirement models, respectively. As forest F is constructed recursively in this manner, the result holds for the complete forest. □

The proof of Theorem 2 follows now.

Proof of Theorem 2. For $CP = (\mathcal{P}, \mathcal{R})$, $CP' = (\mathcal{P}', \mathcal{R}')$ is a partial control problem of CP, denoted by $CP' \preceq CP$, if $\mathcal{P}' \subseteq \mathcal{P}$ and $\mathcal{R}' \subseteq \mathcal{R}$. From Lemma 6 it follows that the forest F constructed from G_{cp} can be analyzed instead of G_{cp} directly. Therefore, it is shown next that no synthesis is needed if (each tree in) the forest is acyclic by induction on the depth of each tree in forest F.

Base case Let subgraph $(\mathcal{P}', \emptyset) \subseteq F$ with $\mathcal{P}' \subseteq \mathcal{P}$ be a tree of depth zero, i.e., it only contains leaf nodes. Then the partial control problem $(\mathcal{P}', \emptyset)$ represented by this subgraph is trivially controllable and nonblocking, and $\mathcal{P}'$ is strongly connected.

Induction hypothesis Assume the set of trees $\{T_1, \ldots, T_k\}$ each with depth at most n such that for each tree $(\mathcal{P}^i, E^i), i \in [1, k]$ the partial control problem $(\mathcal{P}^i, \mathcal{R}^i)$ represented by this subgraph is controllable and nonblocking, and $\mathcal{P}^i \parallel \mathcal{R}^i$ is strongly connected.

Inductive step Denote $\mathcal{P}' = \mathcal{P}^1 \cup \cdots \cup \mathcal{P}^k$ the set of all vertices and $E' = E^1 \cup \cdots \cup E^k$ the set of all edges of the trees with depth at most n, and the control problem $(\mathcal{P}', \mathcal{R}')$ represented by subgraph $(\mathcal{P}', E')$. Let $P \in \mathcal{P} \setminus \mathcal{P}'$ be a vertex not yet in any tree of depth at most n such that for all edges $e \in E$ with $\text{init}(e) = P$, which is assigned to E_P, it holds that $\text{ter}(e) \in \mathcal{P}'$. With other words, P is selected if each of its outgoing edges enter a tree with depth at most n. Note that each edge in E_p has a different terminal vertex, because F is a forest. Let $\mathcal{R} = \{R \in \mathcal{R} \mid event(R) \in \Sigma_P\}$ contain all requirements restricting the behavior of P. It is shown below that the partial control problem represented by tree $(\mathcal{P}' \cup \{P\}, E' \cup E_P)$ of depth at most $n+1$ is controllable and nonblocking, and strongly connected.

From the induction hypothesis it follows that the control problem represented by subgraph $(\mathcal{P}', E')$ is strongly connected, i.e., $\mathcal{P}' \parallel \mathcal{R}'$ is strongly connected. Therefore, similarly to Lemma 3 of Goorden and Fabian (2019), for each requirement $R \in \mathcal{R}$ there exists a string such that state r of $\mathcal{P}'$ is reached that satisfies the condition $cond(R)$, thus enabling controllable event $event(R)$. Analogously to the proof of Lemma 4 of Goorden and Fabian (2019), it holds that a string can be constructed in $\mathcal{P}'$ such that for each path in plant P all controllable events are enabled. Therefore, the partial control problem $(\mathcal{P}' \cup \{P\}, \mathcal{R}' \cup \mathcal{R})$ represented by subgraph $(\mathcal{P}' \cup \{P\}, E' \cup E_P)$ of depth at most $n+1$ is controllable and nonblocking, and for each $P_i \in \mathcal{P}' \cup \{P\}$ all states can be reached from each state. This concludes the inductive step. □

Appendix C. Proof of Theorem 3

Proof of Theorem 3. First, let $\mathcal{P}'$ be the set of all vertices *not* contained in $\mathbb{W}$, i.e., $\mathcal{P}' = \mathcal{P} \setminus (\bigcup_{W \in \mathbb{W}} \bigcup_{V \in W} V)$. From the definition of V it follows that for each vertex $P \in \mathcal{P}'$ there does not exist a path to a cycle. Therefore, the subgraph $(\mathcal{P}', E')$ with $E' = \{e \in E \mid \text{init}(e) \in \mathcal{P}'\}$ is acyclic. From Theorem 2 it directly follows that control problem $(\mathcal{P}', \mathcal{R}')$ represented by this acyclic graph is already controllable and nonblocking, i.e., synthesis can be skipped for this part. Furthermore, from the proof of that theorem it follows that $\mathcal{P}' \parallel \mathcal{R}'$ is also strongly connected.

Now, consider $W \in \mathbb{W}$. The simplified partial control problem $(P_W, \tilde{R}_W)$ represented by $P_W = \bigcup_{V \in W} V$ may contain requirements where the condition is simplified by replacing some state references $P.q$ by $\mathbf{T}$. As $\mathbb{W}$ is the quotient set of $\mathbb{V}$ by $\sim$, it follows from the definition of $\sim$ that each plant P of those replaced state references is from $\mathcal{P}'$. As it is already shown in the previous paragraph that $\mathcal{P}' \parallel \mathcal{R}'$ is strongly connected, it is always possible to reach state $P.q$, which justifies the replacement of this state reference by $\mathbf{T}$. Therefore, if S_W is a controllable, nonblocking, and maximally permissive supervisor of the simplified partial control problem $(P_W, \tilde{R}_W)$, then $\mathcal{P}' \parallel \mathcal{R}' \parallel S_W$ is a controllable, nonblocking, and maximally permissive supervisor for the partial control problem $(\mathcal{P}' \cup P_W, \mathcal{R}' \cup R_W)$, with R_W the non-simplified requirements of $\tilde{R}_W$.

From the definition of $\mathbb{W} = \mathbb{V}/\sim$, it follows that no vertex is shared between two distinct $W_1, W_2 \in \mathbb{W}, W_1 \neq W_2$, i.e., $(\bigcup_{V \in W_1} V) \cap (\bigcup_{V \in W_2} V) = \emptyset$. Let S_{W_1} and S_{W_2} be the controllable, nonblocking, and maximally permissive supervisors for the simplified control problems represented by W_1 and W_2, respectively. Then the supervisors do not share events and it holds trivially that $S_{W_1} \parallel S_{W_2}$ is a controllable, nonblocking, and maximally permissive supervisor.

Finally, combining the above observations for each $W \in \mathbb{W}$ it follows that $\mathcal{P}' \parallel \mathcal{R}' \parallel (\parallel_{W \in \mathbb{W}} S_W) = \mathcal{P} \parallel \mathcal{R} \parallel (\parallel_{W \in \mathbb{W}} S_W)$ is a controllable, nonblocking, and maximally permissive supervisor. □

References

Balemi, S. (1992). *Control of discrete event systems: Theory and application* (Ph.D. thesis), Zurich: Swiss Federal Institue of Technology Zurich.

van Beek, D. A., Fokkink, W. J., Hendriks, D., Hofkamp, A., Markovski, J., van de Mortel-Fronczak, J. M., & Reniers, M. A. (2014). CIF 3: Model-based engineering of supervisory controllers. In *Lecture notes in computer science, Tools and algorithms for the construction and analysis of systems* (pp. 575–580). Springer, Berlin, Heidelberg, http://dx.doi.org/10.1007/978-3-642-54862-8_48.

Brandin, B. A., Malik, R., & Malik, P. (2004). Incremental verification and synthesis of discrete-event systems guided by counter examples. *IEEE Transactions on Control Systems Technology*, *12*(3), 387–401. http://dx.doi.org/10.1109/TCST.2004.824795.

Cassandras, C. G., & Lafortune, S. (2008). *Introduction to discrete event systems* (2nd ed.). Boston: Springer.

Davey, B. A., & Priestley, H. A. (1990). *Introduction to lattice and order*. Cambridge: Cambridge University Press.

Diestel, R. (2017). *Graduate texts in mathematics, Graph theory*. Berlin, Heidelberg: Springer Berlin Heidelberg, http://dx.doi.org/10.1007/978-3-662-53622-3.

Feng, L., & Wonham, W. M. (2006). Computationally efficient supervisor design: Control flow decomposition. In *8th int. workshop on discrete event systems* (pp. 9–14). http://dx.doi.org/10.1109/WODES.2006.1678400.

Flordal, H., & Malik, R. (2009). Compositional verification in supervisory control. *SIAM Journal on Control and Optimization*, *48*(3), 1914–1938. http://dx.doi.org/10.1137/070695526.

Gohari, P., & Wonham, W. M. (2000). On the complexity of supervisory control design in the RW framework. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, *30*(5), 643–652. http://dx.doi.org/10.1109/3477.875441.

Goorden, M. A., & Fabian, M. (2019). No synthesis needed, we are alright already. In *IEEE 15th international conference on automation science and engineering* (pp. 195–202). http://dx.doi.org/10.1109/COASE.2019.8843071.

Goorden, M. A., van de Mortel-Fronczak, J. M., Reniers, M. A., Fokkink, W. J., & Rooda, J. E. (2020). Structuring multilevel discrete-event systems with dependence structure matrices. *IEEE Transactions on Automatic Control*, *65*(4), 1625–1639. http://dx.doi.org/10.1109/TAC.2019.2928119.

Ito, M. (1978). A representation of strongly connected automata and its applications. *Journal of Computer and System Sciences*, *17*(1), 65–80. http://dx.doi.org/10.1016/0022-0000(78)90035-1.

Komenda, J., Masopust, T., & van Schuppen, J. H. (2013). Multilevel coordination control of modular DES. In *52th IEEE conf. on decision and control* (pp. 6323–6328). http://dx.doi.org/10.1109/CDC.2013.6760889.

Komenda, J., Masopust, T., & Schuppen, J. H. v. (2014). Coordination control of discrete-event systems revisited. *Discrete Event Dynamic Systems*, *25*(1), 65–94. http://dx.doi.org/10.1007/s10626-013-0179-x.

Komenda, J., Masopust, T., & van Schuppen, J. H. (2016). Control of an engineering-structured multilevel discrete-event system. In *13th international workshop on discrete event systems* (pp. 103–108). http://dx.doi.org/10.1109/WODES.2016.7497833.

Loose, R., Sanden, B. v. d., Reniers, M., & Schiffelers, R. (2018). Component-wise supervisory controller synthesis in a client/server architecture. *IFAC-PapersOnLine*, *51*(7), 381–387. http://dx.doi.org/10.1016/j.ifacol.2018.06.329.

Ma, C., & Wonham, W. M. (2005). *Lecture notes in control and information sciences: vol. 317, Nonblocking supervisory control of state tree structures*. Springer Berlin Heidelberg.

Malik, R., Åkesson, K., Flordal, H., & Fabian, M. (2017). Supremica–an efficient tool for large-scale discrete event systems. *IFAC-PapersOnLine*, *50*(1), 5794–5799. http://dx.doi.org/10.1016/j.ifacol.2017.08.427.

Markovski, J., Jacobs, K. G. M., van Beek, D. A., Somers, L. J. A. M., & Rooda, J. E. (2010). Coordination of resources using generalized state-based requirements. *IFAC Proceedings Volumes*, *43*(12), 287–292. http://dx.doi.org/10.3182/20100830-3-DE-4013.00048.

Modarres, M. (2016). *Risk analysis in engineering : Techniques, tools, and trends*. CRC Press, http://dx.doi.org/10.1201/b21429.

Mohajerani, S., Malik, R., & Fabian, M. (2014). A framework for compositional synthesis of modular nonblocking supervisors. *IEEE Transactions on Automatic Control*, *59*(1), 150–162. http://dx.doi.org/10.1109/TAC.2013.2283109.

Mohajerani, S., Malik, R., & Fabian, M. (2016). A framework for compositional nonblocking verification of extended finite-state machines. *Discrete Event Dynamic Systems*, *26*(1), 33–84. http://dx.doi.org/10.1007/s10626-015-0217-y.

Moormann, L., Maessen, P., Goorden, M. A., van de Mortel-Fronczak, J. M., & Rooda, J. E. (2020). Design of a tunnel supervisory controller using synthesis-based engineering. In *ITA-AITES world tunnel congress* (pp. 573–578).

de Queiroz, M. H., & Cury, J. E. R. (2000). Modular supervisory control of large scale discrete event systems. In *Discrete event systems* (pp. 103–110). Springer US.

Ramadge, P. J. G., & Wonham, W. M. (1987). Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, *25*(1), 206–230. http://dx.doi.org/10.1137/0325013.

Ramadge, P. J. G., & Wonham, W. M. (1989). The control of discrete event systems. *Proceedings of the IEEE*, *77*(1), 81–98.

Reijnen, F. F. H., Goorden, M. A., van de Mortel-Fronczak, J. M., Reniers, M. A., & Rooda, J. E. (2018). Application of dependency structure matrices and multilevel synthesis to a production line. In *IEEE conference on control technology and applications* (pp. 458–464). http://dx.doi.org/10.1109/CCTA.2018.8511449.

Reijnen, F. F. H., Goorden, M. A., van de Mortel-Fronczak, J. M., & Rooda, J. E. (2017). Supervisory control synthesis for a waterway lock. In *IEEE conference on control technology and applications* (pp. 1562–1568). http://dx.doi.org/10.1109/CCTA.2017.8062679.

Reijnen, F. F. H., Goorden, M. A., van de Mortel-Fronczak, J. M., & Rooda, J. E. (2020). Modeling for supervisor synthesis — a lock-bridge combination case study. *Discrete Event Dynamic Systems*, *30*(3), 499–532. http://dx.doi.org/10.1007/s10626-020-00314-0.

Rijkswaterstaat (2015). *Landelijke brug- en sluisstandaard, vraagspecificatie eisen v4.0: Standard*, Dutch Ministry of Infrastructure and the Environment, In Dutch.

van der Sanden, L. J., Reniers, M. A., Geilen, M. C. W., Basten, A. A., Jacobs, J., Voeten, J. P. M., & Schiffelers, R. R. H. (2015). Modular model-based supervisory controller design for wafer logistics in lithography machines. In *Proc. 18th ACM/IEEE int. conf. on model driven engineering languages and systems* (pp. 416–425). http://dx.doi.org/10.1109/MODELS.2015.7338273.

Stamatis, D. H. (2003). *Texts in theoretical computer science. An EATCS series, Failure mode and effect analysis: FMEA from theory to execution* (2nd ed.). ASQ Quality Press.

Su, R., van Schuppen, J. H., & Rooda, J. E. (2009). Synthesize nonblocking distributed supervisors with coordinators. In *17th mediterranean conference on control and automation* (pp. 1108–1113). http://dx.doi.org/10.1109/MED.2009.5164694.

Su, R., van Schuppen, J. H., & Rooda, J. E. (2010). Aggregative synthesis of distributed supervisors based on automaton abstraction. *IEEE Transactions on Automatic Control*, *55*(7), 1627–1640. http://dx.doi.org/10.1109/TAC.2010.2042342.

Wonham, W. M., & Cai, K. (2019). *Supervisory control of discrete-event systems* (1st ed.). Springer, http://dx.doi.org/10.1007/978-3-319-77452-7.

Wonham, W. M., & Ramadge, P. J. G. (1988). Modular supervisory control of discrete-event systems. *Mathematics of Control, Signals, and Systems*, *1*(1), 13–30.

Zaytoon, J., & Carre-Meneatrier, V. (2001). Synthesis of control implementation for discrete manufacturing systems. *International Journal of Productions Research*, *39*(2), 329–345. http://dx.doi.org/10.1080/00207540010002388.